L Number	Hits	Search Text	DB	Time stamp
1	3124	database near3 version	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:45
2	902	(database near3 version) and hierarch$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:45
3	720	((database near3 version) and hierarch$5) and compar$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:46
4	1	(((database near3 version) and hierarch$5) and compar$3) and "row pointer"	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:47
5	282	(((database near3 version) and hierarch$5) and compar$3) and pointer	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:47
6	140	((((database near3 version) and hierarch$5) and compar$3) and pointer) and commit$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:47
7	90	(((((database near3 version) and hierarch$5) and compar$3) and pointer) and commit$5) and merg$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:47
8	42	((((((database near3 version) and hierarch$5) and compar$3) and pointer) and commit$5) and merg$5) and (update same modif$5)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:48
9	31	(((((((database near3 version) and hierarch$5) and compar$3) and pointer) and commit$5) and merg$5) and (update same modif$5)) and null	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:49
10	19	((((((((database near3 version) and hierarch$5) and compar$3) and pointer) and commit$5) and merg$5) and (update same modif$5)) and null) and snapshot	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:56
11	77	declaring near3 state	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:56
12	0	(((((((((database near3 version) and hierarch$5) and compar$3) and pointer) and commit$5) and merg$5) and (update same modif$5)) and null) and snapshot) and (declaring near3 state)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:56
13	20648	707/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:05
14	4	707/$.ccls. and (declaring near3 state)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:57

15	18	((((((((database near3 version) and hierarch$5) and compar$3) and pointer) and commit$5) and merg$5) and (update same modif$5)) and null) and snapshot) and 707/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:05
16	15	(((((((((database near3 version) and hierarch$5) and compar$3) and pointer) and commit$5) and merg$5) and (update same modif$5)) and null) and snapshot) and 707/$.ccls.) and synchr$6	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:17
17	4569	database same synchr$6	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:17
18	13041	update near3 user	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:18
19	709	(database same synchr$6) and (update near3 user)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:18
20	184	((database same synchr$6) and (update near3 user)) and archiv$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:18
21	166	(((database same synchr$6) and (update near3 user)) and archiv$5) and version	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:18
22	8	((((database same synchr$6) and (update near3 user)) and archiv$5) and version) and (compar$3 same row)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:19
23	2	6560670.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:20
24	3	6216205.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:20
25	0	6216205.pn. and null	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:21
26	1	6560670.pn. and null	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:21
27	0	(6560670.pn. and null) and commit$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:22
28	0	(6560670.pn. and null) and modif$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:22

29	0	(6560670.pn. and null) and update	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:22
30	1	(6560670.pn. and null) and chang$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 15:22
-	13392	optimiz$5 near3 data	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/03 14:45
-	852	(optimiz$5 near3 data) and ((first and second) same row)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/01 14:39
-	250	((optimiz$5 near3 data) and ((first and second) same row)) and pointer	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/01 14:39
-	222	(((optimiz$5 near3 data) and ((first and second) same row)) and pointer) and compar$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/01 14:39
-	97	((((optimiz$5 near3 data) and ((first and second) same row)) and pointer) and compar$5) and null	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/01 14:40
-	97	(((((optimiz$5 near3 data) and ((first and second) same row)) and pointer) and compar$5) and null) and (change or update or delet$3 or add$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/01 14:40
-	79	((((((optimiz$5 near3 data) and ((first and second) same row)) and pointer) and compar$5) and null) and (change or update or delet$3 or add$3)) and equal	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/01 14:41
-	10	(((((((optimiz$5 near3 data) and ((first and second) same row)) and pointer) and compar$5) and null) and (change or update or delet$3 or add$3)) and equal) and (compar$3 near (first or second))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/01 14:42
-	19230	"first row" and "second row"	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:14
-	0	pointer and commit45 and compar$5 and merg$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:14
-	1178	pointer and commit$5 and compar$5 and merg$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:14
-	19	("first row" and "second row") and (pointer and commit$5 and compar$5 and merg$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:15

-	20587	707/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:15
-	3	707/$.ccls. and (pointer and commit$5 and compar$5 and merg$3) and ("first row" and "second row")	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:16
-	9732	microsoft.as.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:16
-	75	microsoft.as. and ("first row" and "second row")	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:16
-	0	(microsoft.as. and ("first row" and "second row")) and (pointer and commit$5 and compar$5 and merg$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:16
-	0	(microsoft.as. and ("first row" and "second row")) and commit$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:16
-	27	(microsoft.as. and ("first row" and "second row")) and updat$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:17
-	573	("first row" and "second row") and null	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:17
-	67	(("first row" and "second row") and null) and commit$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:17
-	22	((("first row" and "second row") and null) and commit$5) and merg$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 11:17
-	20	(((("first row" and "second row") and null) and commit$5) and merg$3) and compar$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 12:46
-	8	(((("first row" and "second row") and null) and commit$5) and merg$3) and (compar$3 same row)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/06/02 12:46

First Hit Fwd Refs

Generate Collection Print

L6: Entry 6 of 17 File: USPT Apr 10, 2001

DOCUMENT-IDENTIFIER: US 6216205 B1
**** See image for Certificate of Correction ****
TITLE: Methods of controlling memory buffers having tri-port cache arrays therein

Application Filing Date (1):
19980521

Brief Summary Text (23):
According to yet another preferred aspect of the present invention, the memory cells in the supplemental memory array may comprise dynamic random access memory (DRAM) cells having very substantially reduced unit cell size compared to the unit cell size of a conventional dual-port buffer memory device (e.g., dual-port FIFO) having dual-port SRAM cells therein. Here, the ratio of the number of memory cells in the supplemental memory array (e.g., DRAM array) relative to the number of potentially significantly larger tri-port cells (e.g., tri-port SRAM cells) is made so high that the total size of the supplemental memory cells and tri-port cells is only slightly greater than that of the supplemental memory array alone. For example, according to the preferred embodiment of the present invention having four (4) tri-port registers, the supplemental memory array may be designed to contain 2048 rows.times.2304 columns 4,718,592 DRAM cells. The tri-port memory array may be designed to contain 4.times.18.times.128=9216 tri-port cells. Thus, even if the unit cell size of a tri-port cell (e.g., SRAM tri-port cell) were to be thirty two (32) times as large as the unit cell size of a single DRAM cell, the total area of DRAM and tri-port cells would only be 6.25% greater than the total area of the DRAM cells alone. Therefore, the present invention may consume a total area which is approximately equal to a conventional DRAM device of equal capacity. However, to the outside world, the present invention provides a sequential buffer memory device with the most preferred features of a current state-of-the-art dual-port-cell based buffer memory device, yet requires only a fraction of the area.

Brief Summary Text (28):
Preferably, the step of determining a next-to-read register as the second free register is closely followed by the step of transferring next-to-read data relative to the data in the current read register from the supplemental memory array to the second free register. Moreover, in the event the memory device contains read row and write row pointers which point to respective rows in the supplemental memory array, the step of performing read arbitration is preferably preceded by the step of updating the read row pointer so that the value of the read row pointer can be used during read arbitration. For example, a step can be performed to compare the values of the read row and write row pointers and then assign one of the four registers in the cache memory array as the next-to-read register based on the outcome of the comparison. This next-to-read register then receives next-to-read data relative to the data in the current read register. Here, the write row pointer is controlled to point to a current write row in the supplemental memory array which is to receive data from the current write register once the current write register has been filled. The read row pointer may also be controlled to point to a row in the supplemental memory array which contains next-to-read data relative to the data in the current read register. Then, during performance of the read arbitration step, the next-to-read register is determined as the current write register if the read row pointer points to the current write row when the step of

comparing the read row and write row pointers is performed.

Detailed Description Text (30):
As best illustrated by FIG. 9, the operations 400 for performing write arbitration include determining a next-to-write register as a first free register in the cache memory array if the current read and write registers are different registers, Blocks 402 and 404, or as the next-to-read register if the current read and write registers are the same register, Blocks 402 and 406. Here, the operations for determining whether the current read and write registers are the same or different at Block 402 are preferably performed by comparing whether the values of the current read register pointer 17a and current write register pointer 19a of FIG. 1 are the same. Alternatively, the values of the read row and write row pointers 25 and 23, respectively, may be compared to determine whether the current read and write registers are the same or different. For example, the read row pointer 25 may be preferably configured to provide a first pointer to the row in supplemental memory 30 containing data in the current read register and a second pointer to the row containing next-to-read data relative to the data in the current read register. Thus, a direct comparison of equivalency between the value of the first pointer and the value of the write row pointer could be made. However, because a comparison of the read and write row pointers may be more computationally complex (e.g., requiring an 11-bit comparison) than a comparison of the values of the current read and current write register pointers (e.g., requiring a 4-bit comparison if each register is assigned one bit entry within a four-bit register), a comparison of the current read and write register pointers is preferred.

Detailed Description Text (31):
As illustrated best by FIG. 10, the operations 500 for performing read arbitration include determining a next-to-read register as the current write register if the current write register contains next-to-read data relative to data in the current read register, Blocks 502 and 504. Read arbitration operations also include determining the next-to-read register as a second free register in the cache memory array 20 if the current read and write registers are different registers and the next-to-read register is not the current write register, Blocks 502, 506 and 510, or as the next-to-write register if the current read and write registers are the same register, Blocks 502, 506 and 508. According to a preferred aspect of the read arbitration operations 500, the operations for determining the next-to-read register as the second free register are closely followed by the step of transferring next-to-read data relative to the data in the current read register from the supplemental memory array 30 to the second free register, Blocks 502, 506, 510 and 512, via the steering circuitry of FIG. 1. As with the write arbitration operations 400, the operations for determining whether the next-to-read register and the current write register are the same or different at Block 502 may be performed by comparing the values of the read row pointer 25 and write row pointer 23 to determine if they point to the same row in the supplemental memory 30. However, the operation for determining whether the current read register and current write register are the same or different at Block 506, is preferably similar to the operation described with respect to Block 402 of FIG. 9. In other words, a comparison of the current read register pointer 17a and current write register pointer 19a may be performed.

Detailed Description Text (32):
Here, the operations for performing read arbitration are preferably preceded by an operation to update the read row pointer 25, as described more fully hereinbelow with respect to FIG. 6., so that an updated value of the read row pointer/counter 25 can be used for comparison with the write row pointer/counter 23. For example, an operation can be performed during read arbitration to compare the value of the read row pointer 25 (which preferably points to a row in the supplemental memory containing next-to-read data relative to the data in the current read register) with the value of the write row pointer 23, and then assign one of the four registers in the cache memory array as the next-to-read register based on the

outcome of the comparison. This assigned next-to-read register then receives next-to-read data relative to the data in the current read register, as illustrated by Block 512 in FIG. 10. Here, the write row pointer 23 is preferably controlled to point to a current write row in the supplemental memory array 30 which is to receive data from the current write register once the current write register has been filled. Then, during performance of the read arbitration operation 500, the next-to-read register is determined as the current write register if the read row pointer 25 points to the current write row when the step of comparing the values of the pointers is performed, Blocks 502 and 504. The next-to-read register is also determined as the next-to-write register at Blocks 506 and 508. if the current read row is the same as the current write row because the current read register pointer 17a and the current write register pointer 19a contain the same value. Alternatively, although less preferred, the read row pointer 25 may also provide a pointer to the current read row (which is typically one row below the next-to-read row in the supplemental memory array 30) and this pointer may be directly compared (e.g., as an 11-bit operand) with the value of the write row pointer 23.

CLAIMS:

5. The method of claim 4, wherein said step of performing read arbitration comprises comparing the read row and write row pointers.

6. The method of claim 5, wherein the write row pointer points to a current write row in the supplemental memory array which is allocated to receive data from the current write register; and wherein said step of performing read arbitration comprises determining the next-to-read register as the current write register if the read row pointer points to the current write row when said step of comparing the read row and write row pointers is performed.

7. The method of claim 2, wherein the memory device contains a current read register pointer which points to the current read register and a current write register pointer which points to the current write register; and wherein said step of performing read arbitration comprises comparing the current read register pointer to the current write register pointer to determine whether the next-to-read register should be assigned as the next-to write register.

8. The method of claim 6, wherein the memory device contains a current read register pointer which points to the current read register and a current write register pointer which points to the current write register; and wherein said step of performing read arbitration comprises comparing the current read register pointer to the current write register pointer to determine whether the next-to-read register should be assigned as the next-to write register.

9. The method of claim 8, wherein said step of performing write arbitration comprises comparing the current read register pointer to the current write register pointer to determine whether the next4o-write register should be assigned as the next-to-read register.

32. The method of claim 27, wherein the memory device contains read row and write row pointers which point to respective rows in the supplemental memory array; wherein said step of performing read arbitration is preceded by the step of updating the read row pointer; and wherein said step of performing read arbitration comprises comparing the values of the read and write row pointers.

33. The method of claim 32, wherein the memory device contains a current read register pointer which points to the current read register and a current write register pointer which points to the current write register; and wherein said step of performing read arbitration comprises comparing the values of the current read register pointer and the current write register pointer.

First Hit Fwd Refs

Generate Collection Print

L6: Entry 2 of 17 File: USPT May 6, 2003

DOCUMENT-IDENTIFIER: US 6560670 B1
TITLE: Inter-row configurability of content addressable memory

Application Filing Date (1):
20000614

Brief Summary Text (4):
A content addressable memory (CAM) system is a storage system that can be instructed to compare a specific pattern of comparand data with data stored in its associative CAM array. The entire CAM array, or segments thereof, is searched in parallel for a match with the comparand data. The CAM device typically includes a priority encoder to translate the highest priority matching location into a match address or CAM index.

Brief Summary Text (5):
The CAM array has rows of CAM cells that each stores a number of bits of a data word. U.S. Pat. No. 5,440,715 describes a technique for expanding the width of the data words beyond that of a single row of CAM cells. Multiple data words can be width expanded together to form a data line. It appears, however, that the CAM system of the '715 patent will not always output the correct match address. For example, assume a first data line of two data words ZY is stored in data words 0 and 1, respectively, and a second data line of two data words WZ is stored in data words 2 and 3, respectively. When a comparand data line of WZ is provided for comparison, the first cycle compare with W will indicate a match with data word 2 only. The second cycle compare with Z will indicate a match with data words 0 and 3 and match lines ML0 and ML3 will be activated. When the priority encoder is enabled, it will output a match address of 0 instead of 3 since ML0 is the highest priority match line.

Brief Summary Text (6):
Additionally, it appears that the CAM system of the '715 patent will not always function correctly when each data line has different numbers of data words. For example, assume that a data line of 5 words VWXYZ is loaded into data word locations 0-4, and a data line of 4 words VWXY is loaded into data word locations 5-8. When a comparand data line of VWXY is provided to the CAM array, ML3 and ML8 will both be activated and the priority encoder will incorrectly output an address of three that stores the last word of a five word data line and not the last word of a four word entry.

Brief Summary Text (12):
A method of searching for a matching data word chain in the CAM array is also disclosed. For one embodiment, the method includes determining the address of a first row in the CAM array that stores data matching a first data word in the data word chain, and determining that a second row in the CAM array stores data that matches a second data word in the data word chain and the address of the first row. The method may further include appending a null pointer to the first data word, and comparing the first data word and the null pointer with the data stored in the CAM array. The method may also include appending the address of the first row to the second data word, and comparing the second data word and the address of the first row with the data stored in the CAM array.

Brief Summary Text (14):
Another embodiment of invalidating one or more valid data words of a data word chain stored in a content addressable memory (CAM) array is also disclosed. The data word chain comprises a sequence of n data words each stored in a unique row of CAM cells of the CAM array, where n is an integer greater than one, and wherein each data word has an associated pointer stored in the same row as the corresponding data word. The method includes (a) determining that the CAM array stores the data word chain; (b) comparing the address of the row in the CAM array that stores the nth data word with the pointers of each of the data words; (c) setting x equal to the pointer of the nth data word; (d) setting y equal to the address of the row in the CAM array that stores the nth data word; (e) invalidating the data word at address y; (f) comparing x with the pointers of the remaining valid data words; (g) setting y equal to x; (h) setting x equal to the pointer at the address of x in the CAM array; and (i) repeating (e)-(h), inclusive, until x is equal to a null pointer associated with the first data word of the data word chain. The method may further include invalidating the first data word of the data word chain.

Detailed Description Text (4):
FIG. 1 is a block diagram of one embodiment of a CAM system 100 that includes a CAM array 102, priority encoder 104, write circuitry 106, address decoder 108, comparand register 110, and one or more mask registers 112. Other CAM systems may be used. CAM system 100 may also include other well-known circuits including an instruction decoder or control logic, read circuitry, and flag logic for generating a match flag, full flag, and multiple match flag. CAM array 102 has n rows of CAM cells 116(0)-116(n-1), where n is any integer, that may be any type of CAM cells including binary or ternary CAM cells. Each CAM row includes a pointer field 114 and a data word field 112. The data word fields each store a data word in a data word chain that includes a sequence of one or more data words. The pointer fields each store a pointer for the associated data word of the same row. The first data word of each data word chain is assigned a null pointer that is a predetermined number. For other embodiments, the last data word may be assigned the null pointer. The null pointer may be a number that is greater than n so that it is outside the addressable range of rows of the CAM array. For another embodiment, a separate bit or bits may be used to indicate the null pointer. All data words other than the first data word in a particular data word chain are assigned a pointer that is the address or index of the previous data word in the chain. In this way, the data words in a data word chain are linked together regardless of where in the CAM array each of the data words is stored.

Detailed Description Text (6):
A search key or comparand data may be provided to the CAM array for simultaneous comparison with the pointers and/or the data words. The comparand data is provided on the CBUS and may be stored in comparand register 110. The results of a comparison operation are indicated on match lines ML(0)-ML(n-1) and provided to priority encoder 104. The priority encoder determines the index or address of the highest priority matching entry and outputs the index to the RBUS. Alternatively, priority encoder 104 may determine the index of the lowest priority matching entry, or use any other priority determination scheme.

Detailed Description Text (10):
An example of searching for a data word chain that matches comparand data is shown in FIG. 4. First, at step 402, the null pointer is appended to the first data word of the data word chain to be searched. The first data word and the null pointer are then provided as comparand data and compared with the entries in the CAM array. If there is no match, the process stops at step 414 and a no match condition may be signaled (e.g., by disabling a match flag). If there is a match, the index or address of the row in the CAM array at which the first data word is stored is output to the RBUS via priority encoder 104 (step 406). If there are no more data

words in the data word chain (step 408), the process stops at step 414. If, however, there is another data word in the data word chain to be searched on, the index associated with the first data word is appended as the pointer for the next data word (step 410), and the next data word and its pointer are then provided as comparand data to the CAM array and searched against all of the entries therein (step 412). If there is a match, then the process returns to step 406 and continues until all of the data words of a data word chain are searched for in the CAM array. When there is no match at step 412, the process stops at step 414.

Detailed Description Text (11):
The search process of FIG. 4 may be further illustrated by searching for data word chain ABC in CAM array 102 of FIG. 2. First, the null pointer 1024 is appended to the first data word A (step 402). Data word A and the null pointer are then simultaneously searched against all entries in the CAM array (step 404). A match is determined at row 0 and the index 0 output to the RBUS (step 406). The index 0 is appended to the next data word B (steps 408 and 410), and the concatenation compared against the entries in the CAM array (step 412). A match is determined at row 1 and the index 1 output to the RBUS. The index 1 is appended to the next data word C, and the concatenation compared against the entries in the CAM array. A match is determined at row 2 and the index 2 output to the RBUS. Since there are no more data words to be searched, the process stops and indicates a match condition (e.g., by enabling a match flag). The index of the last data word C output to the RBUS may then be used, for example, to access data stored in another memory.

Detailed Description Text (16):
FIG. 8 is one embodiment of determining the existing tree structure of data word chains in a CAM array before writing one or more data words of a new data word chain to the array. Steps 802-808 determine the extent to which any stored data word chains have common initial data words with a new data word chain. First, at step 802, the first data word of the new entry and a null pointer are provided as comparand data and compared with all entries in the CAM array. If no match is found, then the null pointer is appended to the new first data word and the new data word chain and its corresponding pointers are loaded into available rows in the CAM array (step 810) using the process, for example, illustrated in FIG. 9. If, however, a match is found at step 802, then a data word chain already stored in the CAM array has a common first data word with the new data word. The new data word chain does not need to store its first data word and null pointer in the CAM array as this entry already exists. The index or address of the first data word stored in the CAM array is read from the RBUS and appended to the next data word in the new data word chain as its pointer (step 804). The next data word and its pointer are then provided as comparand data and compared with all entries in the CAM array (step 806). If there is no match, the branch in the tree structure has been determined and the balance of the data words in the new chain are written into available rows in the CAM array (step 812) as illustrated, for example, in FIG. 9. If there is a match at step 806, then the searched data word does not need to be written to the CAM array since it already exists in the CAM array with the correct relationship to the first data word. If there are no more data words in the new chain, the process stops at step 814; however, if there are more data words, the process continues until a mismatch condition occurs or there are no more data words in the new chain.

Detailed Description Text (18):
The process of FIGS. 8 and 9 can be further illustrated by writing the tree structure of FIG. 6 into the CAM array as shown in FIG. 7. For purposes of illustration, CAM array 102 is empty when data word chains ABCD and ABDE are provided for storage. First, chain ABCD is written into CAM array 102. At step 802, the null pointer 1024 is appended to data word A and compared with the entries in the CAM array. Since no entries are found, data word A and the null pointer are provided to write circuitry 106 at step 902 of FIG. 9. Since the array is empty, NFA and X are equal to 0 (step 904). Data word A and the null pointer are then

written into row 0 (step 906), and index 0 is appended to data word B (step 910). X is then updated to the next free address of 1 (step 906). Data word B and its pointer 0 are then written into row 1 (step 906), and index 1 is appended to data word C (step 910). X is then updated to the next free address of 2 (step 906). Similarly, data word C and its pointer 1 are then written into row 2 (step 906), and index 2 is appended to data word D (step 910). X is then updated to the next free address of 3 (step 906). Finally, data word D and its pointer 2 are then written into row 3 (step 906) and the process stops (steps 908 and 912).

Detailed Description Text (19):
After chain ABCD is written into the array, chain ABDE is written into the array using the processes of FIGS. 8 and 9. At step 802, the null pointer 1024 is appended to data word A and the two are compared with the entries in the array. A match is determined with the entry that is already stored at address 0. Thus, data word A is not written again into the array. The index 0 is output to the RBUS by priority encoder 104 and appended to the next data word B (step 804). Data word B and its pointer 0 are compared with the entries in the array (step 806) and a match is determined with the entry that is already stored at row 1. Thus, data word B is not written again into the array. The index 1 is output to the RBUS and appended to the next data word D (step 804). Data word D and its pointer 1 are compared with the entries in the array (step 806) and no match is determined. Thus, the sequence AB was previously stored in the array, but the sequence ABD was not. The process then transitions to step 902 of FIG. 9. At step 902, data word D and its pointer 1 are provided to the write circuitry for writing into the array. At step 904, the next free address of 4 in the CAM array is stored in X. Data word D and its pointer 1 are then written into row 4 (step 906), and index 4 is appended to data word E (step 910). X is then updated to the next free address 5 in the CAM array (step 906). Data word E and its pointer 4 are then written into row 5 (step 906) and the process stops (steps 908 and 912).

Detailed Description Text (22):
FIG. 10 is one embodiment of deleting or invalidating a data word chain from the array. The process will be described to delete chain ABCD from the array of FIG. 7. Initially, the process determines whether the data word chain exists in the array by performing a search operation (step 1002) and recording the pointer and index associated with each data word in the chain. This may be accomplished by using the process of FIG. 4 and recording the pointer and index associated with each data word A, B, C, and D of the chain. The index 3 of the last data word D is then compared with the pointers while the data words are masked by mask register(s) 112. If there is a match, then another, longer data word chain includes chain ABCD (e.g., if another data word chain of ABCDE was also stored in the array), and data word D will not be deleted and the process stops at step 1020. In this example, the index 3 does not match any of the pointers and variable X is set to data word D's pointer value of 2, and Y is set to the index value of 3 (step 1006). Data word D is then deleted (step 1008). X is then compared with the pointers while the data words are masked (step 1010). Since data word D and its pointer were already deleted or invalidated, there should be no match between the pointers and X. If, however, there is a match, then the process stops at step 1020 and an error condition may be signaled. Y is then updated with X (step 1012), and X is set to the pointer at an index of X (step 1014). That is, X is set to the pointer of the next data word C up in the chain. For this example, the pointer 1 at address 2 associated with data word C is loaded into X. Since X is not the null pointer (step 1016), data word C is deleted (step 1008). X is then compared with the pointers while the data words are masked (step 1010). Since data word C and its pointer were already deleted or invalidated, its data pointer will not generate a match. The pointer 1 associated with data word D of chain ABDE, however, will register a match indicating that a branch in the tree has been identified. As such, the process stops at step 1020 and does not delete data words A or B. Thus, data words C and D have been deleted, but data words A and B have remained valid in the array for chain ABDE. For an alternative embodiment in which chain ABDE did not exist in the

o

array, then the process of FIG. 10 would continue through step 1016 until it is determined that X is the null pointer and the first data is then deleted at step 1018.

CLAIMS:

1. A method of determining that a data word chain matches data stored in a a plurality of rows of content addressable memory (CAM) cells of a CAM array, wherein the data word chain comprises a sequence of at least two data words, the method comprising: determining the address of a first row in the CAM array that stores (i) data matching a first data word in the data word chain and (ii) a first pointer value; and determining that a second row in the CAM array stores (i) data that matches a second data word in the data word chain and (ii) a second pointer value that corresponds to the address of the first row, the second pointer value being different from the first pointer value.

2. The method of claim 1, wherein determining the address of the first row comprises comparing the first data word with data stored in the CAM array.

3. The method of claim 2, wherein determining the address of the first row comprises: appending a null pointer value to the first data word; and comparing the first data word and the null pointer value with the data and first pointer value stored in the first row of the CAM array.

5. The method of claim 1, wherein determining that the second row stores data that matches the second data word and the second pointer value comprises: appending the address of the first row to the second data word; and comparing the second data word and the address of the first row with the data and second pointer value stored in the second row of the CAM array.

8. A method of determining that a data word chain matches data stored in a plurality of rows of content addressable memory (CAM) cells of a CAM array, wherein the data word chain comprises a sequence of at least two data words, the method comprising: comparing a first data word and a corresponding null pointer with the data stored in the CAM array; determining that the first data word and the null pointer match data stored in a first row of the CAM array; determining the address of the first row of the CAM array; appending the address of the first row to a second data word; comparing the second data word and the address of the first row with data stored in the CAM array; and determining that the second data word and the address of the first row match data stored in a second row of the CAM array.

12. A method of writing a data word chain into a plurality of rows of content addressable memory (CAM) cells of a CAM array, wherein the data word chain comprises a sequence of at least two data words, the method comprising: determining the address of a first row in the CAM array that does not store valid data; writing a first data word of the data word chain and a first pointer value to the first row in the CAM array; and writing a second data word of the data word chain and a second pointer value to a second row in the CAM array, the second pointer value corresponding to the address of the first row and being different from the first pointer value.

13. The method of claim 12, further comprising determining the address of the second row before writing the second data word and the second pointer value to the second row.

14. The method of claim 12, wherein writing the first data word and the first pointer value to the first row in the CAM array comprises: appending a null pointer to the first data word; and writing the null pointer and the first data word to the first row in the CAM array.

33. A method of invalidating one or more valid data words of a data word chain stored in a content addressable memory (CAM) array, wherein the data word chain comprises a sequence of n data words each stored in a unique row of CAM cells of the CAM array, where n is an integer greater than one, and wherein each data word has an associated pointer stored in the same row as the corresponding data word, the method comprising: (a) determining that the CAM array stores the data word chain; (b) comparing the address of the row in the CAM array that stores the nth data word with the pointers of each of the data words; (c) setting x equal to the pointer of the nth data word; (d) setting y equal to the address of the row in the CAM array that stores the nth data word; (e) invalidating the data word at address y; (f) comparing x with the pointers of the remaining valid data words; (g) setting y equal to x; (h) setting x equal to the pointer at the address of x in the CAM array; and (i) repeating (e)-(h), inclusive, until x is equal to a null pointer associated with the first data word of the data word chain.

35. A method of invalidating one or more valid data words of a data word chain stored in a content addressable memory (CAM) array, wherein the data word chain comprises a sequence of n data words each stored in a unique row of CAM cells of the CAM array, where n is an integer greater than one, and wherein each data word has an associated pointer stored in the same row as the corresponding data word, the method comprising: (a) determining that the CAM array stores the data word chain; (b) comparing the address of the row in the CAM array that stores the nth data word with the pointers of each of the data words; (c) setting x equal to the pointer of the nth data word; (d) setting y equal to the address of the row in the CAM array that stores the nth data word; (e) marking the data word at address y for invalidation; (f) comparing x with the pointers of the remaining valid data words; (g) setting y equal to x; (h) setting x equal to the pointer at the address of x in the CAM array; (i) repeating (e)-(h), inclusive, until x is equal to a null pointer associated with the first data word of the data word chain; and (j) invalidating all marked data words.

36. A method of invalidating one or more valid data words of a data word chain stored in a content addressable memory (CAM) array, wherein the data word chain comprises a sequence of n data words each stored in a unique row of CAM cells of the CAM array, where n is an integer greater than one, and wherein each data word has an associated pointer stored in the same row as the corresponding data word, the method comprising: (a) determining that the CAM array stores the data word chain; (b) comparing the address of the row in the CAM array that stores the nth data word with the pointers of each of the data words; (c) setting x equal to the pointer of the nth data word; (d) setting y equal to the address of the row in the CAM array that stores the nth data word; (e) marking the data word at address y for invalidation; (f) comparing x with the pointers of the remaining valid data words; (g) setting y equal to x; (h) setting x equal to the pointer at the address of x in the CAM array; (i) repeating (e)-(h), inclusive, until x the result of (f) indicates more than one match; and (j) invalidating all marked data words.

42. A method of writing a data word chain into an array of content addressable memory (CAM) cells within a CAM device, the method comprising: storing a first data word of the data word chain in a data field of a first row of the CAM cells; storing a null pointer value in a pointer field of the first row of the CAM cells; and storing a second data word of the data word chain in a data field of a second row of the CAM cells.

43. The method of claim 42 further comprising storing an address of the first row of CAM cells in a pointer field of the second row of the CAM cells.